Pascal–2 V2.1/RT-11 User Guide

Introduction to the User Guide

This is the introductory section, the User Guide. It explains:

- How to compile and run Pascal programs;
- How to interpret program listings and error messages;
- Some details of the compilation process.

This guide assumes that you are familiar with:

- Simple RT-11 commands;
- A text editor (e.g., EDIT, TECO, KED);
- Elementary Pascal programming.

This guide is not:

- An introduction to Pascal (see *Programming in Pascal* by Peter Grogono);
- A detailed description of Pascal–2 (see the Language Specification, and Doug Cooper's *Standard Pascal User Reference Manual*);
- An expert's guide to Pascal–2 (see the Programmer's Guide).

Getting Started

The first step in running a Pascal program is to enter the program into the computer and store it in the file system. Use a familiar text editor to enter the program; store it in a file with the extension .PAS. The Pascal–2 compiler accepts free-format program files, so use blanks, tabs, new lines, and form feeds as desired to help make the program readable.

This Pascal version of a program is called the source program, or the source file. All other versions of the program are translations from the source program.

Compiling the Program

After editing, you must compile the program—translate it into a form that the computer can execute—and link it to the Pascal–2 support library. With the compiler and the support library on the system disk and with a source file called TEST.PAS, the entire compilation process follows this example:

```
.R PASCAL
*TEST

.LINK TEST,SY:PASCAL
```

As the example shows, the .PAS extension may be omitted from file names on commands to the Pascal–2 system but must be included in commands to other RT-11 systems such as the editor.

To illustrate the compilation process, let's say that the program

```
program First (output);
begin
  write ('"Things are best in their beginnings"');
  writeln (' -- Blaise Pascal');
end.
```

is stored in the file FIRST.PAS.

Compilation proceeds as follows:

```
.R PASCAL
*FIRST

LINK FIRST.SY:PASCAL
.RUN FIRST
"Things are best in their beginnings" -- Blaise Pascal
```

Notice that no errors were detected. The next example shows what happens if detectable errors are present in the source program.

Checking For Errors

The Pascal-2 compiler detects nearly 150 types of "grammatical" errors in a program: errors in syntax such as missing semicolons, undefined identifiers, missing **begin** and **end** reserved words, and similar mistakes. As an example, the following program contains a deliberate error: a semicolon is missing between the program heading and the reserved word **begin**.

```
program Second (output)
begin
  writeln ('Things get worse as they continue');
end.
```

Semicolon errors (the most common errors made by beginning Pascal programmers) are always detected by the compiler:

```
.R PASCAL
*SECOND
Pascal-2 RT11 SJ V2.1D   9-Feb-84   7:08 AM     Site #1-1     Page 1-1
Oregon Software, 6915 SW Macadam Ave., Portland, Oregon 97219, (503) 245-2202
SECOND

    1       program Second (output)
                                   ^19
*** 19: Use ';' to separate statements


 *** There was 1 line with errors detected ***
?Errors detected: 1
```

For each detected error, a line of the source program is printed, then an arrow indicating the approximate position of the error, then a message describing the error. (The number "19" is the error message number generated by the compiler.) See Appendix A of the Programmer's Guide for a complete list of detectable compilation errors.

Errors Detected at Run-Time

The errors discussed so far have been compilation errors — errors detected by the compiler. Run-time errors, on the other hand, occur when a program is executing, after it has been compiled and linked.

A run-time error such as "array subscript out of bounds" stops the program at the point of error. The Pascal–2 error reporting system prints header information and the error message, then traces the program's execution history, procedure by procedure, from the point of error back to the main program. The error traceback, or "walkback," is intended to make debugging easier by showing precisely where the program stopped and which procedures were called to reach that point.

The following is an example of a run-time error and procedure walkback. (The program has already been compiled and linked.) Line numbers appearing in the walkback correspond to line numbers in the source listing, not line numbers in individual procedures.

```
.R PASCAL
*CUSTOM

PASCAL--Fatal error at user PC= 7244B
Array subscript out of bounds

Error occurred at line 64 in procedure writelastname
Last called from line 90 in procedure buildcustomerfile
Last called from line 103 in program customs
```

The first line of the walkback contains header information about the program and gives the type of error (fatal or I/O) and the program counter at the time of the error.

The second line of the walkback is the error message issued by the error reporting system. Appendix B of the Programmer's Guide contains a list of run-time errors and a short explanation of what may have gone wrong. (If the error is I/O-related, an additional line is printed that provides the system I/O error code and the name of the file causing the error.)

The third line shows the location of the error in terms of source program lines. The remaining lines of the walkback indicate the reverse order in which the procedures were called.

Pascal–2 also includes the capability to trap and recover from run-time I/O errors and to print additional information about the error. See the Programmer's Guide for discussion of these more advanced techniques.

Compilation Options

The Program Listing

Many times, to correct an error, you need to see more of the program than just the line on which the error appears. The Pascal–2 compiler can be directed to display the entire program, with all detected errors and other information. This is the "listing" of the program.

To obtain a listing file (.LST), include the **list** switch in the compilation command line:

```
.R PASCAL
*SECOND/LIST
```

To get a program listing at a terminal, specify TT: as the listing file, as shown below. The listing also may be written to the line printer or a disk file.

```
.R PASCAL
*THIRD,TT: = THIRD/LIST
Pascal-2 RT11 SJ V2.1A   5-Aug-83   7:04 PM     Site #1-1     Page 1-1
Oregon Software, 2340 SW Canyon Road, Portland, Oregon 97201, (503) 226-7760
THIRD,TT: = THIRD/LIST

         1       program Third (output)
                                       ^19
*** 19: Use ';' to separate statements
         2       begin
         3         writeln ('Things get hazy if you stare at them');
         4       end.

***There was 1 line with errors detected ***
```

The listing is printed in pages, with a heading on each page showing the program name, the exact version of the Pascal–2 compiler, the date and time, and the licensed user identification. The listing also prints out, in the left-hand column, the line number for each line of the program. You also may use the **errors** switch to create a listing file containing only the lines with detected errors.

As illustrated in the above example of **list**, a compilation switch modifies the compilation process in some way. A switch is signified by a slash and a descriptive name. The Programmer's Guide describes all of the compilation switches, but the next examples show the most commonly used ones. The examples also demonstrate some of the features of the Pascal–2 package — the Debugger, the Profiler, and the PASMAT formatter, and double-precision real number format.

The Formatter

Suppose you have a program, EFACT.PAS, that calculates an approximation of **e**, the base of the natural logarithms, by summing the series

$$1 + 1/1! + 1/2! + 1/3! + ... + 1/N!$$

until additional terms do not affect the approximation.

Remember that the compiler will accept a program in whatever format you choose. So the program may look like this:

```
program Efact(output);
var E, Delta, Fact: real;
N: integer;
begin
E:=1.0; N:=1; Fact:=1.0; Delta:=1.0;
repeat
E:=E+Delta;
N:=N+1; Fact:=Fact*N; Delta:=1/Fact;
until E = (E+Delta);
write('With ', n:1, ' terms, ');
writeln('the value of e is',E:18:15);
end.
```

To make the program more readable, you decide to format the program with PASMAT, one of the Pascal–2 utility programs. Give the following command:

```
.R PASMAT
*EFACT
```

and the program is reformatted to look like this:

```
program Efact(output);

  var
    E, Delta, Fact: real;
    N: integer;

  begin
    E := 1.0;
    N := 1;
    Fact := 1.0;
    Delta := 1.0;
    repeat
      E := E + Delta;
      N := N + 1;
      Fact := Fact * N;
      Delta := 1 / Fact;
    until E = (E + Delta);
    write('With ', n:1, ' terms, ');
    writeln('the value of e is',E:18:15);
  end.
```

(PASMAT has many other formatting options. See the Utilities Guide for details.) Now proceed to compile the program.

```
.R PASCAL
*EFACT

.LINK EFACT,SY:PASCAL
.RUN EFACT
With 11 terms, the value of e is 2.718282000000000
```

The Debugger

Even after you have corrected any syntax errors caught by the compiler, the program may still yield unexpected results. In this situation, Pascal–2's interactive Debugger can help uncover and correct the problems. The Debugger takes control of the program and responds to your commands, displaying execution information in a Pascal context. With the Debugger, you can watch the progress of the computation, and you can display intermediate values without making any program changes. You can then spot the point at which values go awry and correct the error.

To do this, use the **debug** switch to compile the program with the Debugger. (In most cases, you probably also will want to overlay the Debugger module. See the Debugger Guide for details.)

First, compile and link the program with the commands:

```
.R PASCAL
*EFACT/DEBUG

.LINK EFACT,SY:PASCAL
```

The **debug** compilation produces four output files: EFACT.LST, EFACT.SYM, EFACT.SMP, and EFACT.OBJ. You need the listing file to determine the places to set breakpoints in the program. Don't worry about the other three output files, but don't delete them or the listing file. The Debugger uses all of them.

After doing a **debug** compilation, you will find it handy to have a printout of the listing file. The file will look like this:

```
Pascal-2 RT11 SJ V2.1A   5-Aug-83   7:04 PM      Site #1-1      Page 1-1
Oregon Software, 2340 SW Canyon Road, Portland, Oregon 97201, (503) 226-7760
EFACT/DEBUG

 Line Stmt
    1       program Efact(output);
    2
    3         var
    4           E, Delta, Fact: real;
    5           N: integer;
    6
    7    1    begin
    8    2      E := 1.0;
    9    3      N := 1;
   10    4      Fact := 1.0;
   11    5      Delta := 1.0;
   12    6      repeat
   13    7        E := E + Delta;
   14    8        N := N + 1;
   15    9        Fact := Fact * N;
   16   10        Delta := 1 / Fact;
   17           until E = (E + Delta);
   18   11      write('With ', n:1, ' terms, ');
   19   12      writeln('the value of e is',E:18:15);
   20         end.

*** No lines with errors detected ***
```

Two columns of numbers appear on the left side of each page. The first column, labeled **Line**, numbers each line of the source program. The second column, labeled **Stmt**, gives the statement

number of the first statement on that line. The statement numbers start at 1 for each procedure or function, increasing by one as each statement is compiled. The Debugger uses these statement numbers to identify breakpoint locations in the compiled program.

In the program **Efact**, for instance, you may want to set a breakpoint at statement number 7. This is the point at which the approximation of **e** changes. If the program compiles correctly but produces unsatisfactory results, you may set the breakpoint at **MAIN,7** to monitor the approximation to **e** as the program runs. We'll do just that in the next example.

Notice that the Debugger allows you to set the breakpoints. In this example, you tell the program to write the value of **e** at the breakpoint and then continue. (See the Debugger Guide for details on these commands.)

```
.RUN EFACT

Pascal Debugger V3.00 -- 12-Aug-83

Debugging program: EFACT

} B(MAIN,7) <W(E);C>  ———————— at breakpoint, write E and continue
} G  ———————————————————————————————— start program
Breakpoint at MAIN,7  E := E + Delta;
1.0000000E+00
Breakpoint at MAIN,7  E := E + Delta;
2.0000000E+00
Breakpoint at MAIN,7  E := E + Delta;
2.5000000E+00
Breakpoint at MAIN,7  E := E + Delta;
2.6666667E+00
Breakpoint at MAIN,7  E := E + Delta;
2.7083335E+00
Breakpoint at MAIN,7  E := E + Delta;
2.7166669E+00
Breakpoint at MAIN,7  E := E + Delta;
2.7180557E+00
Breakpoint at MAIN,7  E := E + Delta;
2.7182541E+00
Breakpoint at MAIN,7  E := E + Delta;
2.7182789E+00
Breakpoint at MAIN,7  E := E + Delta;
2.7182817E+00
With 11 terms the value of e is 2.718282000000000

Program terminated.

Breakpoint at MAIN,12  writeln('the value of e is', E: 18: 15);
} Q  ———————————————————————————————— quit
```

Double Precision

The computed value in the previous examples is printed with 7 significant digits. You may need greater precision for some programs. To get extended precision, use the **double** switch, which computes to 15 significant digits. The **double** switch allows you to print a more precise value for **e**. (See the Programmer's Guide for details.)

```
.R PASCAL
*EFACT/DOUBLE

.LINK EFACT,SY:PASCAL
.RUN EFACT
With 19 terms, the value of e is 2.718281828459050
```

The Profiler

Finally, let's examine the program for efficiency by using the **profile** switch and by adding the **PRFILE** module to the Linker input. "Profiling" shows the number of times each statement is executed, giving you the opportunity to optimize sections of code that are executed many times.

The Profiler takes control of your program and asks for the name of the profile output file. The default extension is .PRO.

```
.R PASCAL
*EFACT/PROFILE

.LINK EFACT,SY:PRFILE,SY:PASCAL
.RUN EFACT

profile  V2.1  12-Aug-83

Profiling program: EFACT

Profile output file name? EFACT  ——— Output goes to default extension
With 11 terms, the value of e is 2.718282000000000

Program terminated.

Profile being generated
```

The output file looks like this:

```
Pascal-2 RT11 SJ V2.1A   5-Aug-83   7:04 PM      Site #1-1      Page 1-1
Oregon Software, 2340 SW Canyon Road, Portland, Oregon 97201, (503) 226-7760
        EFACT/PROFILE

          Line Stmt
            1      program Efact(output);
            2        var
            3          E, Delta, Fact: real;
            4          N: integer;
            5
     1      6    1   begin
     1      7    2     E := 1.0;
     1      8    3     N := 1;
     1      9    4     Fact := 1.0;
     1     10    5     Delta := 1.0;
    10     11    6     repeat
    10     12    7       E := E + Delta;
    10     13    8       N := N + 1;
    10     14    9       Fact := Fact * N;
    10     15   10       Delta := 1 / Fact;
           16            until E = (E + Delta);
     1     17   11     write('With ', n: 1, ' terms ');
     1     18   12     writeln('the value of e is', e: 18: 15);
           19         end.

      *** No lines with errors detected ***

                    PROCEDURE EXECUTION SUMMARY

Procedure name     statements   times called   statements executed

MAIN                   12            1             57 100.00%

There are 12 statements in 1 procedures in this program.
      57 statements were executed during the profile.
```

The leftmost column of the profile listing shows the number of times each line is executed. The Profiler listing concludes with a "Procedure Execution Summary" that details each procedure name, the number of times it is called, the number of statements it contains, and the number of statements it executes. Note, too, that the summary shows the percent of execution count taken by each program block. (In this example, with only one procedure, the portion is 100%.) Given this information, you can attempt to optimize the procedures and statements that use a disproportionately large part of the time ("90 percent of the time on 10 percent of the program").

See the Profiler section of the Debugger Guide for more information and for a much more detailed example.

Your Next Step

Thus ends your guided tour through Pascal–2. At this point, you should be able to run a few simple programs. Before getting into complex programs, however, you should consult the Programmer's Guide, the Language Specification, and the Debugger Guide.